Exercise 2 Report

Intelligent Signal Processing - Final Assessment

Overview

This report focuses on discussing the idea behind the methodologies used to implement audio file compression solutions. The implementation steps are explained in detail using markdown cells in the submitted Jupyter notebook.

Main Application

As directed in the task sheet, the Rice Coding algorithm was implemented to build a lossless compression algorithm to compress audio files.

Rice Coding Theory

The Rice coding algorithm, also known as Golomb-Rice coding is a simple data compression algorithm, most commonly used in image compression, often as a component of more advanced compression methods like the JPEG and HEVC (High-Efficiency Video Coding) standards. Its primary use lies in encoding residuals, which represent the data that isn't accounted for by the primary compression algorithm.

The core idea behind Rice coding is to divide the integer values into a quotient and a remainder using a value M where $M = 2^k$. The quotient is then encoded using a unary code (a sequence of repeated 1s followed by a 0), and the remainder is encoded using binary representation. The combination of the unary code of the quotient followed by the binary code of the remained yields the final Rice code.

The parameter K determines the efficiency of the coding. Smaller values of K result in more compact coding for small values but might lead to longer codes for larger values. Larger values of K make the encoding of large values efficient but can lead to longer codes for small values. Hence, it is usually efficient for encoding integer values that follow a certain distribution of values, where the data contains a large number of integers that yield smaller codes for the given value of K.

Results

The results of the Rice Coding algorithm implementation were not good. While the solution works as expected when encoding and decoding, the encoded file is larger than the original files by quite a large amount.

Table 1 presents the results of using the Rice Coding algorithm on the provided audio files.

File Name	K	Original Size (Bytes)	Compressed Size (Bytes)	Compression %
Sound1.wav	2	1002088	4115719	410.71%
Sound1.wav	4	1002088	1516266	151.31%
Sound2.wav	2	1008044	4348596	431.39%
Sound2.wav	4	1008044	1575348	156.28%

Table 1: Rice coding algorithm compression results.

Analysis

It can be observed from Table 1, that Rice Coding is performed very poorly in terms of compression since it ended up increasing the size of the file by a large margin. This is mainly due to the larger byte values in the byte array of the files. The results are better with higher values of K, where the best results are when $K = 7$. But even then, the size of the resulting compressed file is approximately 110% of the original file.

Further Development

To further develop the compression algorithm and improve compression, the Lempel–Ziv–Storer–Szymanski (LZSS) algorithm was implemented.

LZSS Theory

The LZSS algorithm is a lossless data compression algorithm that was derived from the LZ77 algorithm. It's designed to achieve compression by replacing repeated occurrences of data in the input stream with references to earlier occurrences, thereby reducing redundancy and saving space. Hence, it tends to work well with data that has patterns in it.

Results

The results of the LZSS algorithm were promising and the algorithm was able to reduce the size of one of the files. The implementations of both the encode and decode functions for the algorithm work as expected.

Table 2 presents the results of using the LZSS algorithm on the provided audio files using different window sizes.

File Name	Window Size	Original Size (Bytes)	Compressed Size (Bytes)	Compression %
Sound1.wav	64	1002088	958299	95.63
Sound1.wav	128	1002088	932663	93.07
Sound2.wav	64	1008044	1134040	112.50
Sound2.wav	128	1008044	1134038	112.50

Table 2: LZSS algorithm compression results.

Analysis

Looking at the results in the table, it can be seen that the LZSS algorithm does a significantly better job at compression compared to the Rice coding algorithm. However, it still fails to effectively compress the `Sound2.wav` file. This can most probably be attributed to the algorithm not being able to find as many patterns in `Sound2.wav` as it did in `Sound1.wav`.

Another way to improve the performance of the algorithm would be to increase the window size for pattern detection. This could help with compression, but it will also slow down the algorithm.